# 基于 S3C6410 平台的嵌入式 Linux 系统 LCD 驱动模块

黄相平，余水宝，夏 灿

（浙江师范大学 数理与信息工程学院，浙江 金华 321004）

**摘 要：** 以 S3C6410 处理器为核心，采用嵌入式 Linux 操作系统，通过帧缓冲设备的驱动方式，实现了 TFT 液晶显示器的驱动程序。经过测试字体的放大与旋转及图形的显示，证明该驱动程序稳定，具有很高的实用价值。

**关键词：** 嵌入式 Linux；帧缓冲；TFT 驱动程序；Platform 机制

**中图分类号：** TP311 **文献标识码：** A **文章编号：** 1674-7720(2013)13-0009-04

## LCD driver module based on S3C6410 platform of embedded Linux

Huang Xiangping, Yu Shuibao, Xia Can

(College of Mathematics, Physics and Information Engineering, Zhejiang Normal University, Jinhua 321004, China)

**Abstract:** In this paper, we use S3C6410 processor and the embedded Linux operating system, through the FrameBuffer device, implements the TFT LCD driver. Through display font rotation, font zoom and graphics, proved that the driver is stable and has the very high practical value.

**Key words:** embedded Linux; FramBuffer; TFT driver; Platform mechanism

由于 Linux 具有开放源码、易于移植、内核可裁剪、资源丰富、免费等优点，使它在嵌入式领域越来越流行。随着手持设备的与日俱增，人机交互界面向着更方便、更直观的方向发展。触摸屏的应用让数据的显示和输入结合为一体，使得人机交互界面更简单、友好，已经广泛应用于嵌入式系统当中。在实际工程中，LCD 驱动与触摸驱动是分开的，所以本文只涉及 LCD 驱动。由于 ARM 核广泛地应用于嵌入式系统中，所以选用的是基于 ARM1176JZF-S 核的 S3C6410 CPU，其频率可以达到 667 MHz，对多媒体的处理速度很快。

液晶显示器相对于传统的 CRT 有很多优点：轻薄、能耗低、辐射少等，市场的占有率也越来越大。LCD 有多种类型，比如 TN、STN、TFT、LTPS 等[1]。TFT 型 LCD 中，晶体管矩阵依显示信号开启或关闭液晶分子的电压，使液晶分子轴转向而成“亮”或“暗”的对比，避免了 TN 与 STN 依靠电场效应而造成响应时间长的缺点，而且播放动画或视频不会有 STN 的拖影现象[2]。背光灯的应用，使其在日光下也能显示图像，应用广泛。LTPS 是 TFT 衍生的新一代技术，价格较贵，所以这里是关于 TFT 液晶显示器的驱动。

## 1 设备参数及硬件连接

### 1.1 TFT 液晶显示屏参数

本文选用的是万鑫的 4.3 寸的 TFT 真彩屏，分辨率为 480×272，数据格式采用 24 bit RGB 接口模式，其主要参数如表 1 所示，参数名后面跟的英文字符串表示在程序中使用的宏。

表 1 LCD 主要技术参数

| 参数名 | 参数值(典型值) | 单位 |
|---|---|---|
| 时钟 CLK | 9 | MHz |
| 水平分辨率 HOZVAL | 480 | CLK |
| 行显示前延 HFPD | 2 | CLK |
| 行显示后延 HBPD | 2 | CLK |
| 水平同步宽度 HSPW | 41 | CLK |
| 垂直分辨率 LINEVAL | 272 | CLK |
| 垂直显示前延 VFPD | 2 | CLK |
| 垂直显示后延 VBPD | 2 | CLK |
| 垂直同步宽度 VSPW | 10 | CLK |

### 1.2 TFT 与 S3C6410 的硬件连接

一块 LCD 屏显示图像不仅需要 LCD 驱动器，还需要 LCD 控制器。通常 LCD 驱动器会与 LCD 玻璃基板制作在一起，而 S3C6410 内部集成了一个 LCD 控制器，所

以 LCD 与 6410 之间可以直接连接。图 1 显示了 LCD 液晶显示器与 LCD 插槽之间的连接，图 2 显示了 LCD 插槽与 S3C6410 之间的连接。VD [0:23] 是 GRB 数据线，VDD_LCD 接的是 LCD 的电源，I2CSDA0 背光，VDEN 表示数据可以传输颜色数据，其他的与表 1 的名称对应，其中 HSYNC、VSYNC、VDEN、VCLK 分别与 LHSYNC、LVSYNC、LVDEN、LVCLK 连接。

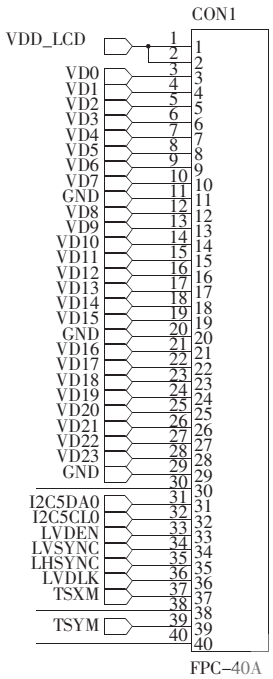


图 1 TFT 液晶显示器与 LCD 插槽的连接


图 2 LCD 插槽与 S3C6410 之间的连接

## 2 Platform 总线下的 LCD 驱动

### 2.1 Platform 总线简介

在 Linux2.6 后的设备驱动模型中，总线、设备和驱动这三个实体称之为总线设备驱动模型，即为同类的设备设计了一个框架，而框架的核心层实现了该类设备通用的一些功能，而不用程序员自己再去实现；驱动与设备相分离，使得编写主机控制器驱动和外设驱动并行，它们之间不再互相关联，实现分层与分离的思想，从而提高驱动的可移植性。总线将设备和驱动绑定。在系统每注册一个设备的时候，会寻找与之匹配的驱动；同样地，在注册一个驱动的时候，会寻找与之匹配的设备。匹配是通过比较设备和驱动中的名字来确定，匹配工作由总线完成；一般情况下，驱动会使用设备的资源，所以设备常常要比驱动先注册，而且设备一般会放在板文件中。

在嵌入式系统中，SoC 系统中集成的独立的外设控制器，挂接在 SoC 内存空间的外设不依附于任何总线。在这种情形下，Linux 发明了一种虚拟的总线，就是 Platform 总线，相应的设备称为 platform_device，而驱动为 platform_driver。而 LCD 正是挂接在 Platform 总线下的一个驱动，LCD 驱动先构造一个 platform_device 结构体，然后注册；接着 LCD 驱动构造一个 platform_driver 结构体，并注意 platform_driver 中的名字要与 platform_device 中的名字相同，这样当注册 platform_driver 时就能够匹配设备，Platform 总线就会调用 platform_driver 中的 probe 函数，这个函数是 LCD 驱动真正注册的函数；同理，卸载函数为 platform_driver 中的 remove 函数。

### 2.2 LCD 驱动的 platform_device 与 platform_driver 的注册

#### 2.2.1 分配初始化一个 platform_device 结构体

使用 platform_dev_lcd43=platform_device_alloc("lcd43",-1) 分配并设置一个 platform_device，第一个参数就是 platform_device 的名字，第二个参数为 -1 表示设备只有一个。在相应的移除函数中调用 platform_device_release(platform_dev_lcd43) 函数释放分配的内存，这两个函数都是内核自带的函数。下文中，如果没有特别介绍，都是内核自带的函数。在这里，为了避免在 platform_driver 的注册中引入太多的内核函数，并没有把资源放入 platform_device 中，而是要用时再在 platform_driver 中实现，这里只是简单地演示实际工程中这种 platform_device 结构的使用；实际工作中，只需把 platform_driver 的资源放入 platform_device 的 resource 中，然后调用 platform_get_resource() 函数来取得相应的资源就可以了。

接下来调用 platform_device_register(platform_dev_ lcd43) 注册 platform_device，在相应的移除函数中调用 platform_device_unregister(platform_dev_lcd43)。

#### 2.2.2 定义初始化一个 platform_driver 结构体

platform_driver 结构体就是 LCD 驱动的结构体，把 platform_driver 中的成员初始化注册后，内核就可以调用这个驱动程序了。

```
static struct platform_driver platform_lcd43_driver={
.probe=lcd43_probe,
.remove=lcd43_remove,
.driver={
    .name="lcd43",
    .owner=THIS_MODULE,
                        /* 内核的宏，表示当前模块 */
    }
};
```

接下来调用 platform_driver_register (&platform_lcd43_driver) 注册 platform_driver 结构体。

相应的移除函数为 platform_driver_unregister (&platform_lcd43_driver)。当调用 platform_driver_register() 函数后，内核就自动的去找跟它相匹配的 platform_device，如果找到，就调用 platform_driver 的 probe 函数。

## 3 LCD 驱动的编写

### 3.1 帧缓冲的概念

帧缓冲是 Linux 系统为显示设备提供的一个接口，它将显示缓冲区进行抽象，屏蔽了图像硬件的底层差异，允许上层应用程序在图形模式下直接对缓冲区进行

读写。用户只要在显示缓冲区中与显示点对应的区域写入颜色值，对应的颜色会自动在屏幕上以对应的点显示。帧缓冲驱动的应用非常广泛，Qt/Embedded、X Window、MiniGUI 都是利用帧缓冲进行图像的绘制。LCD 驱动程序的设计就是写一个帧缓冲驱动程序，它的主设备号为 29，当注册一个帧缓冲设备时，会自动地在/dev 目录下创建一个 fbn 设备文件，$n$ 为注册前系统帧缓冲设备的个数，当系统中没有帧缓冲设备时，$n$ 为 0。

### 3.2 帧缓冲设备 probe 函数的编写

#### 3.2.1 帧缓冲结构体的初始化

帧缓冲设备驱动的编写工作主要集中在 platform_driver 中的 probe 函数的编写，probe()函数完成 LCD 驱动程序真正要做的事情，它需要完成：

(1)申请一个 struct fb_info 结构体，根据 LCD 屏参数并初始化。帧缓冲设备最关键的一个数据结构体是 fb_info 结构体，它包含帧缓冲设备属性和操作的完整描述。

```
fb43_info=framebuffer_alloc(0,NULL);
                        /* 申请一个 struct fb_info 结构体 */
fb43_info->screen_size=HOZVAL*LINEVAL*4;
                        /* 设置帧缓冲区的大小 */
```

fb_info 结构体成员 fb_fix_screeninfo 结构的初始化，fb_fix_screeninfo 记录用户不可修改的显示器的参数，如屏幕大小、缓冲区地址等。

```
strcpy(fix43->id,"lcd_4.3");            /* 设置名字 */
fix43->line_length=HOZVAL*4;
   /* 一行像素占的字节，1 像素 4 字节，高 8 位丢弃 */
fix43->smem_len=LINEVAL*HOZVAL*4;
                        /* 设置帧缓冲内存的大小 */
fix43->type=FB_TYPE_PACKED_PIXELS;
                        /* 设置 LCD 的型号 */
fix43->visual=FB_VISUAL_TRUECOLOR;
                        /* 设置显示为真彩色 */
```

fb_info 结构体成员 fb_var_screeninfo 结构的初始化，fb_var_screeninfo 记录用户可以更改的显示器参数，包括屏幕分辨率和每像素点的比特数等。

```
var43->xres=HOZVAL;
                /*x 方向实际像素个数，表 1 中的数值 */
var43->yres=LINEVAL;
                /*y 方向实际像素个数，表 1 中的数值 */
var43->xres_virtual=HOZVAL;
                /*x 方向虚拟像素个数，没实现虚拟 */
var43->yres_virtual=LINEVAL;
                /*y 方向虚拟像素个数，没实现虚拟 */
var43->xoffset=0; /*x 方向实际与虚拟间的偏移像素 */
var43->yoffset=0; /*y 方向实际与虚拟间的偏移像素 */
var43->bits_per_pixel=32;
        /* 每像素 32 位表示，实际只是 24 位的 RGB*/
var43->red.length=8;                    /* 红占 8 位 */
var43->red.offset=16;                   /* 红偏移 16 位 */
var43->green.length=8;                  /* 绿占 8 位 */
var43->green.offset=8;                  /* 绿偏移 8 位 */
var43->blue.length=8;                   /* 蓝占 8 位 */
var43->blue.offset=0;                   /* 蓝偏移 0 位 */
var43->hsync_len=HSPW;
        /* 以下都是表 1 中的数据，硬件相关 */
var43->vsync_len=VSPW;
var43->right_margin=HBPD;
var43->left_margin=HFPD;
var43->lower_margin=VBPD;
var43->upper_margin=VFPD;
clk43=clk_get(NULL,"hclk");
hclkval=clk_get_rate(clk43);
    /* 获得系统的 HCLK，LCD 的时钟由它分频得到 */
var43->pixclock=hclkval/(hclkval/9000000);
/* 设置像素时钟，注意 C 语言会对整型变量除法运算 */
```

(2)完成 S3C6410 的 LCD 控制器的初始化

S3C6410 自带 LCD 控制器，只要把 LCD 控制寄存器设置合适的参数，S3C6410 会自动发出控制 LCD 的时序，而不需要程序员再去关心时序的问题。这部分的设置大都是硬件相关，不同的 CPU 设置一般不会一样。在 Linux 系统中，CPU 不能直接使用物理地址，所以需要先经过 ioremap()函数建立新页表，使物理地址映射到逻辑地址[3]。后文中出现的与具体寄存器相关的指针变量都是经过 ioremap()重新映射后的地址。

```
mifpcon=ioremap(0x7410800C,4);
spcon=ioremap(0x7F0081A0,4);
*mifpcon&=~(1<<3);
*spcon&=~(3);
*spcon|=1;
```

根据 S3C6410 手册要求的设置，要使用 LCD 控制器，必须把 MOFPCON 寄存器的第三位置零，SPCON[1:0]位要为“01”，表示为 RGB 模式。

```
regs43->vidcon0=((hclkval/9000000-1)<<6)|(1<<4);
                        /* 设置时钟源及分频系数 */
regs43->vidcon1=(1<<6)|(1<<5);
/* 因为 S3C6410 的行列同步脉冲与 TFT 屏的相反，所以要设置这两个脉冲反转，要根据具体的 TFT 屏芯片手册设置 */
    /* 设置垂直延时参数，LCD 硬件相关，见表 1 内容 */
regs43->vidtcon0=((VBPD-1)<<16)|((VFPD-1)<<8)|(VSPW-1)<<0;
    /* 设置垂直延时参数，LCD 硬件相关，见表 1 内容 */
regs43->vidtcon1=((HBPD-1)<<16)|((HFPD-1)<<8)|(HSPW-1)<<0;
regs43->vidtcon2=((LINEVAL-1)<<11)|((HOZVAL-1)<<0);
                        /* 设置行与列的分辨率 */
```

```
regs43->wincon0&=~(0xf<<2);
regs43->wincon0|=(0xb<<2);
                        /* 设置为 24 位 RGB 格式 */
regs43->vidosd0a=(0<<11)|(0<<0);
                        /* 设置左上角开始值,不留黑框 */
regs43->vidosd0b=((HOZVAL-1)<<11)|((LINEVAL-1)<<0);
                        /* 设置右下角位置 */
regs43->vidosd0c=HOZVAL*LINEVAL;
                        /* 设置显示框的大小,以字为单位 */
regs43->vidw00add0b0=fb43_info->fix.smem_start;
                        /* 设置物理地址 */
regs43->vidw00add1b0 =(fb43_info->fix.smem_start + LINEVAL*HOZVAL*4)&0xffffff;
                        /* 设置帧缓冲内存结束地址的低 24 位 */
```

(3)分配帧缓冲区的内存,同时设置帧缓冲区的虚拟及物理地址。

```
fb43_info->screen_base =dma_alloc_writecombine (NULL, fb43_info->screen_size,
&fb43_info->fix.smem_start, GFP_KERNEL);
                        /* 设置帧缓冲区内存,虚拟,物理地址 */
```

分配帧缓冲内存,设置固定参数的物理地址以及fb_nfo结构的虚拟地址,第二个参数为帧缓冲区的大小,第三个参数会返回申请的帧缓冲区的开始物理地址,设置为在固定参数中没有设置的物理地址,函数返回值为帧缓冲区的虚拟地址,最后的一个参数为分配内存的标志,设置为最常用的GFP_KERNEL。

(4)开LCD时钟、电源、背光,并注册帧缓冲设备驱动。函数定义如下:

```
clk_enable_43lcd();
/* 打开 LCD 的时钟,因为 S3C6410 可以关闭一些外设来省电 */
get_43lcd_ctr(fb43_info, hclkval);
                        /* 完成 S3C6410 的 LCD 控制器的初始化 */
fb43_info->fbops=&fb43_ops;
                        /* 设置帧缓冲的操作函数 */
fb43_info->pseudo_palette=&fb43_pseudo_palette;
                        /* 设置调色板 */
enable_43lcd();         /* 开 LCD 电源、背光 */
```

### 3.2.2 设置LCD与S3C6410的连接设置

从连接图可以看出,LCD的数据线主要是在通用输入输出I口和J口[4]。由6410的芯片手册如图3所示,I口与J口基本一样。只需要把它们的控制寄存器设置为LCD模式即可,这时S3C6410会根据LCD控制器的一些数据,在恰当的时间发出颜色数据。设置如下:

```
*gpicon=0xaaaaaaaa;
*gpjcon=0xaaaaaaaa;
```

| GPICON | Bit | Description | | Initial State |
|---|---|---|---|---|
| GPI0 | [1:0] | 00=Input<br>10=LCD VC[0] | 01=Output<br>11=reserved | 00 |

图3 GPICON控制寄存器的配置描述

### 3.2.3 开LCD电源及背光,注册帧缓冲设备

```
*gpbcon&=~(0xf<<24);
*gpbcon|=(0x1<<24);
                        /* 设置 GPB 的第 6 位为输出模式 */
*gpbdat|=(1<<6);        /* 开启背光 */
*gpfcon&=~(3<<28);
*gpfcon|=(1<<28);
                        /* 设置 GPF 的第 14 位为输出模式 */
*gpfdat|=(1<<14);       /* 开启 LCD 电源 */
regs43->vidcon0|=3;     /* 开 LCD 显示 */
regs43->wincon0|=1;
/* 开窗口 0 显示,S3C6410 可以同时显示 5 个窗口,本文选择的是窗口 0*/
register_framebuffer(fb43_info);    /* 注册帧缓冲设备 */
```

### 3.2.4 填充fb_ops结构体

struct fb_info结构中的fb_ops结构体是操纵帧缓冲设备的一些函数的集合,系统调用最后通过它们与LCD控制器硬件打交道。对于其中的fb_fillrect()、fb_copyarea()以及fb_imageblit()成员直接使用通用的cfb_fillrect()、cfb_copyarea()以及cfb_imageblit()就可以了。只需要设置以下几个成员,其他的都用系统默认的函数,LCD驱动就能正确的工作了。

```
struct fb_ops fb43_ops={
    .owner=THIS_MODULE,         /* 表示为当前模块 */
    .fb_fillrect=cfb_fillrect,
    .fb_copyarea=cfb_copyarea,
    .fb_imageblit=cfb_imageblit,
    .fb_setcolreg=fb43_setcolreg,
};
```

fb_setcolreg成员是实现伪颜色表和颜色表的填充,虽然使用24 bit RGB颜色格式不需要伪颜色表,但不实现它会出现一些问题。使用伪颜色表(也称调色板)时,发出的颜色值并不是真正要显示的值,而是在一个颜色表中的索引。当使用大于16 bit的颜色表时,颜色表占据的内存过大,所以使用它不划算。它的设置如下,分别取出RGB的索引值,取其低16 bit,接着根据当前颜色在RGB格式中的偏移值,把其他颜色位置零,最后再把取出的3种颜色合成一个颜色值。设置如下:

```
colorg&=0xffff;
colorg>>=16-bf->length;/*bf->length 是颜色占据的长度, 假如是 16 位 RGB:5:6:5 格式, 红色就占据最高的 5 位,所以低 11 位的值不要 */
colorg<<bf->offset;
                /* 如上,红色占最高 5 位,所以要右移 11 位 */
.....           /* 设置其他两种颜色值 */
return colorr|colorg|colorb;
```

## 4 LCD驱动的测试

用arm-linux-gcc在交叉编译平台上编译后,把生成

的模块文件 lcd.ko 用 insmod 加载到 S3C6410 开发板。把编译好的测试文件也拷入到开发板的文件系统中，运行测试文件[5]，如图 4 所示用 freetype 字形引擎显示的放大及旋转的文字，字体大小为 40 像素，旋转 45°角。如图 5 所示为 libjpeg 库显示的图片，显示的图片为图 4 缩小到 1/8 时的情形（图形上边没有显示是因为 libjpeg 库只能把图片缩小为 1/2、1/4、1/8）。可以看见 LCD 都能正常工作。

本文详述了一种 TFT 液晶显示屏在嵌入式 Linux 平台开发的方法，实现了其字体及图形的稳定显示，证明了该方法能够在一定条件下，LCD 显示性能的满足。

图 4 显示旋转字体

图 5 显示图片

参考文献

[1] 杜春雷.ARM 体系结构与编程[M].北京：清华大学出版社，2003.

[2] 宋宝华.Linux 设备驱动开发详解（第 2 版）[M].北京：人民邮电出版社，2010.

[3] JONATHAN CORBET.Linux.设备驱动程序（第 3 版）[M].北京：中国电力出版社，2006.

[4] STEVENS W R.Unix 环境高级编程（第 2 版）[M].北京：人民邮电出版社，2006.

[5] 韦东山.嵌入式 Linux 应用开发完全手册[M].北京：人民邮电出版社，2008.

（收稿日期：2013-04-10）

作者简介：

黄相平，男，1988 年生，硕士，主要研究方向：嵌入式系统。

余水宝，男，1954 年生，教授，主要研究方向：单片机与嵌入式系统应用，信号检测与处理。

夏灿，女，1988 年生，硕士研究生，主要研究方向：嵌入式系统。